A theory of desynchronisable closed loop systems

Harsh Beohar
Formal methods group
Department of Mathematics and Computer Science
Eindhoven university of technology, The Netherlands
H.Beohar@tue.nl

Pieter Cuijpers
Formal methods group
Department of Mathematics and Computer Science
Eindhoven university of technology, The Netherlands
P.J.L.Cuijpers@tue.nl

The task of implementing a supervisory controller is non-trivial, even though different theories exist that allow automatic synthesis of these controllers in the form of automata. One of the reasons for this discord is due to the asynchronous interaction between a plant and its controller in implementations, whereas the existing supervisory control theories assume synchronous interaction. As a consequence the implementation suffer from the so-called inexact synchronisation problem. In this paper we address the issue of inexact synchronisation in a process algebraic setting, by solving a more general problem of refinement. We construct an asynchronous closed loop system by introducing a communication medium in a given synchronous closed loop system. Our goal is to find sufficient conditions under which a synchronous closed loop system is branching bisimilar to its corresponding asynchronous closed loop system.

1 Introduction

The task of implementing a supervisory controller is non-trivial, even though different theories exist that allow automatic synthesis of these controllers in the form of automata. One of the reasons for this discord is due to the asynchronous interaction between a plant and its controller in implementations, whereas the existing supervisory control theories assume synchronous interaction. We elaborate on this mismatch by first introducing some terminology that is often used in supervisory control theory [14].

Supervisory control theory provides an automatic synthesis of a supervisor that controls a plant in such a way that a corresponding requirement (legal behaviour) is achieved. In supervisory control theory terminology,

- the model that is to be controlled is known as *plant*,
- the model that specifies the requirement is known as *specification*,
- the model that forces the plant to meet the specification by interacting with it is known as *supervisor* or *controller*.
- the interaction between a plant and its supervisor is known as *closed-loop behavior*.

The closed loop behaviour in supervisory control theory is realized by synchronous parallel composition. Informally, it allows a plant and a supervisor to synchronise on common events while other events can happen independently.

One of the main drawbacks while implementing the interaction between a plant and its supervisor, synthesised by supervisory control theory, is inexact synchronization [7]. In practical industrial applications, the interaction between a plant and its supervisor is not synchronous but rather asynchronous. Due to the synchronous parallel composition used in supervisory control theory, the interaction between a plant and its supervisor is strict. By strict, we mean that either plant or supervisor has to wait for

Bliudze, Bruni, Grohmann, Silva (Eds.):
Third Interaction and Concurrency Experience (ICE 2010)
EPTCS 38, 2010, pp. 99–114, doi:10.4204/EPTCS.38.10

the other party while synchronising. To overcome this problem it is important to study asynchronous communication between a plant and its supervisor where communications are delayed in buffers.

Balemi was the first to consider the inexact synchronisation problem, and the solutions given in his PhD thesis [4] were in the domain of automata theory. In [4], an *input-output* interpretation was given between a plant and its supervisor and a special delay operator was introduced to model the delay in communication between the plant and the supervisor. Moreover, for this setup the existence of a supervisor in the presence of delays was also shown in [4]. It was required that the output actions from a plant can occur asynchronously, while the output actions from a supervisor must occur synchronously [18]. In [18] this requirement was relaxed. Furthermore, necessary and sufficient conditions were also provided for the existence of a controller under bounded delay between a plant and its supervisor.

The solutions provided in [4, 18] construct a new supervisor under the presence of bounded delay, which is a computationally expensive procedure. To circumvent this, we present sufficient conditions on a synchronous closed loop system under which the asynchronous closed loop system constructed from it, is a refinement of the given synchronous closed loop system. Moreover, the technique developed in this paper is independent of the size of buffers used. However, we do not analyse the computational complexities associated with the sufficient conditions presented in this paper.

In this paper, we reformulate the inexact synchronisation problem as a problem of refinement in the process algebra TCP [3]. The synchronous closed loop system can be considered as a specification with the asynchronous closed loop system as its implementation. If the given synchronous closed loop system and its corresponding asynchronous closed loop system are branching bisimilar [16], then the asynchronous closed loop system is said to be a refinement of its corresponding synchronous closed loop system. Note that we do not compute an additional supervisor under the presence of delays, instead we assume a given plant and its supervisor. Thus, we solve a refinement problem instead of solving a control synthesis problem.

In the past, the idea of solving a refinement problem was studied [8, 10, 11], but different setups (in comparison with the current paper) were used in these studies. These studies were motivated by the so-called "Foam-rubber wrapper" principle [15], borrowed from the field of delay insensitive circuits. Mathematically, it states that "a process and the same process connected with buffers are equivalent". In [8], the foam-rubber wrapper principle was also studied in the context of the parallel composition and it was shown that an extra condition is required to preserve this principle. In brief, we have a different architecture for the asynchronous closed loop system in this paper and we study the components in the asynchronous closed loop system conjointly, in order to capture desynchronisability.

1.1 Architecture

This paper is a result of the pre-study carried out in [6], where four construction methods are proposed to construct an asynchronous closed loop system from its corresponding synchronous one. In this subsection, we introduce the architecture of an asynchronous closed loop system, discuss the reasonability of using a bag as a buffer and describe one of the abstraction schemes that will be used throughout this paper. We elucidate on these points in the upcoming paragraphs.

An asynchronous closed loop system can be constructed by introducing a buffer between a plant and its supervisor in order to decouple the synchronisation of events between the two. In practice, the buffering mechanism is realised by the interactions of different layers (also known as protocol stack) as shown in Figure 1. In theory, various authors [8, 10, 12] have abstracted from the interaction of different layers by using data structures based on a particular level of abstraction. For example, to model delay insensitive (DI) circuits, which are at a lower level of abstraction (physical layer), wires are used as a

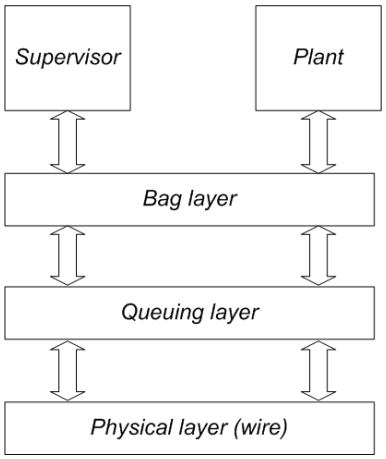


Figure 1: Asynchronous closed loop system in practice.

buffering mechanism [11]. On the other hand to model data flow networks, which are at a higher level of abstraction (in comparison to DI circuits), queues are used as a buffering mechanism [10]. In this paper, we are interested in studying the asynchronous interaction in a closed loop system at an even higher level of abstraction by having a unique queue for every message. Thus, a queue stores only one type of unique message and all queues are allowed to run concurrently without interacting with one another. Such interleaving queues are equivalent to a bag modulo strong bisimulation. Hence, we use a bag as the buffering mechanism in this paper.

It is obvious that upon introduction of the bag as a buffer, the asynchronous closed loop system contains interactions that are not present in the synchronous closed loop system. However, to relate these two closed loop systems by a branching bisimulation relation [16], it is necessary to hide some interactions or define a suitable abstraction scheme. In principle, a synchronous closed loop system can be converted into an asynchronous closed loop system by introducing bags with the following abstraction schemes:

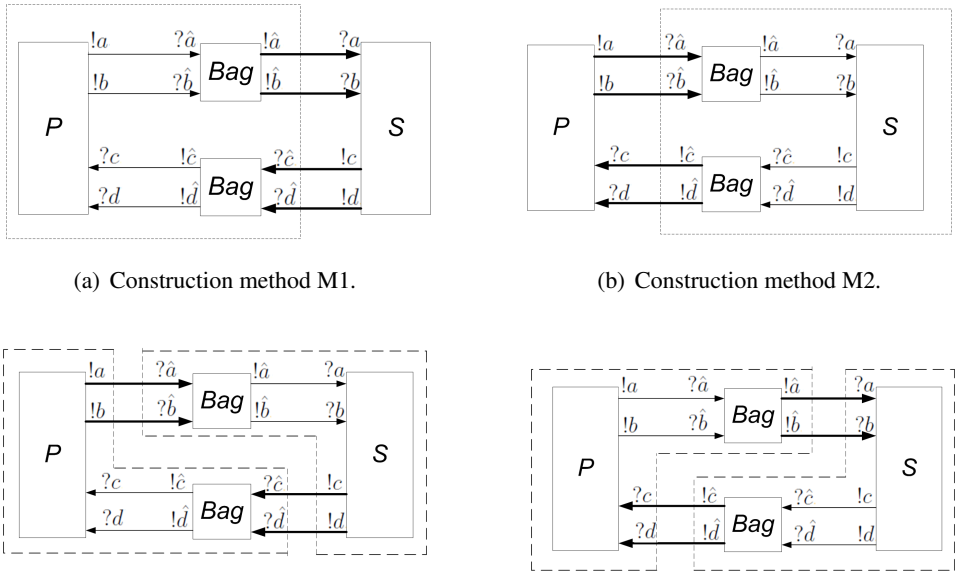
M1. by introducing bags between a plant and its supervisor such that *the interaction between plant and bag is hidden* (see Figure 2(a)).

M2. by introducing bags between a plant and its supervisor such that *the interaction between supervisor and bag is hidden* (see Figure 2(b)).

M3. by introducing bags between a plant and its supervisor such that *the communication among the input actions of both plant and supervisor with bags are hidden* (see Figure 2(c)).

M4. by introducing bags between a plant and its supervisor such that *the communication among the output actions of both plant and supervisor with bags are hidden* (see Figure 2(d)).

In Figure 2, thick lines are used to show the visible interaction and thin lines are used to show the invisible interaction. The notation $!a$ means 'send action a' and $?a$ means 'receive action a'. In this paper, we develop the theory for the construction method M1 (see Section 4 for the rationale behind this choice) and leave other construction methods as open for future study. Moreover, the techniques presented in this paper are restricted to reactive systems (so, no termination).



(a) Construction method M1.

(b) Construction method M2.

(c) Construction method M3.

(d) Construction method M4.

Figure 2: Different ways to construct an asynchronous closed loop system.

1.2 Outline

The remainder of this paper is organized as follows. In Section 2, we start our exposition by defining the overall background required for this paper. Section 3 provides a brief introduction to supervisory control theory with respect to our setup. In Section 4, the construction method M1 is defined formally with its abstraction scheme. In Section 5, we give the formal definition of a desynchronisable closed loop system with the conditions that are sufficient for desynchronisability. Finally, in Section 6 we present the conclusions and propose some directions for future research.

2 Background

In this section, we define the basic notations and definitions that will be used throughout this article. Let Act be a set of action names. We use symbols $a,b,c,\ldots$ to range over the set Act. Then we define the following actions for an action label $a \in Act$,

- $!a$: send action label a.
- $?a$: receive action label a.
- $⁉a$: communicated action label a.

Let A denote the set of all possible actions that are defined as, $A = \{!a, ?a, ⁉a\}_{a\in Act}$. The variables $x,y,z,\ldots$ are used to denote elements from set A when the information about the type of action is irrelevant. The set of all process terms (denoted by $\mathbb{P}$) is then defined by the following grammar. The constant $\mathbf{0}$ is a process term that cannot perform any action, i.e. it can only *deadlock*. A unary operator $x._-$ for each action $x \in A$ is introduced in the TCP syntax, denoting an *action prefix*. Intuitively, the process term $x.p$ performs the action x and then behaves as the process p. The binary operator $+$ denotes the *alternative*

composition or choice between any two processes. The encapsulation operator $\partial_H(\_)$, *blocks execution of actions* from H while enduring the execution of other actions from $A \setminus H$. The abstraction operator $\tau_I(\_)$ *renames* the actions from I to τ, and leaves other actions unchanged.

$\mathbb{P}$	$::=$	$\mathbf{0}$	deadlock process
	$\mid$	$x.\mathbb{P}$	action prefix
	$\mid$	$\mathbb{P}+\mathbb{P}$	alternative composition
	$\mid$	$\mathbb{P} \parallel_\gamma \mathbb{P}$	parallel composition
	$\mid$	$\partial_H(\mathbb{P})$	action encapsulation, where $H \subseteq A$
	$\mid$	$\tau_I(\mathbb{P})$	abstraction (hiding of actions), where $I \subseteq A$
	$\mid$	$\mathscr{R}$	recursive definition

In the remainder of this paper, we assume that the symbols $P,R,S,p,p',s,s'\ldots$ range over the set $\mathbb{P}$. We fix the capital letters P,R,S for processes associated with supervisory control theory. Note that we also use the alphabet operator α and renaming operator ρ from TCP algebra for technical reasons, but not for modeling purposes. The empty process $\mathbf{1}$ is not defined because we are interested in modeling only reactive systems. The notation $\mathscr{R}$ denotes a recursion definition by a set of pairs $\{X_0 = t_0, \ldots, X_m = t_m\}$ where X_i denotes a recursion variable and t_i the process term defining it. The parallel composition operator is parameterized with a communication function $\gamma : A \times A \to A$ such that $\gamma(?a,!a) = \gamma(!a,?a) = \mathbf{?}a$.

The semantic domain of the process terms is a transition system (See [3] for details), which is achieved by the so-called SOS rules [13]. For the sake of completeness, we give the SOS rules of the operators used here in the Appendix A.

Definition 2.1. A *transition system* over a set of actions A is a set Q of states, equipped with a transition relation $\to \subseteq Q \times A \cup \{\tau\} \times Q$. The action $\tau \notin A$ denotes the invisible action. In the semantics of TCP, Q is usually taken to be the set of process terms, i.e., $Q = \mathbb{P}$, and the initial state of a process is defined as the process term itself. □

As mentioned in the introduction, we use branching bisimulation to relate a synchronous closed loop system and its corresponding asynchronous closed loop system in which τ actions are present. The presence of τ actions in an asynchronous closed loop system will become evident in Section 4. We write the transitive closure of the transition relation $\to$ as $\twoheadrightarrow$. The symbol $\equiv$ is used to denote syntactical equivalence between process terms. The shorthand notation $q \xrightarrow{\tau^*}\!\!\!\!\!\rightarrow\, q'$ is defined as $q \equiv q_0 \xrightarrow{\tau} \ldots \xrightarrow{\tau} q_n \equiv q'$ for all $q_i \in Q$ with $i \in [0,n], n \geq 0$.

Definition 2.2. A binary relation $\Phi \subseteq Q \times Q$ is called a *branching bisimulation relation* [3, 16] iff:

- $\forall q,q_1,q',x.\Big[(q,q') \in \Phi \wedge q \xrightarrow{x} q_1 \Rightarrow \exists q'_1,q'_2.\big[q' \xrightarrow{\tau^*}\!\!\!\!\!\rightarrow\, q'_1 \xrightarrow{x} q'_2 \wedge (q,q'_1) \in \Phi \wedge (q_1,q'_2) \in \Phi\big]\Big]$.
- $\forall q,q_1,q'.\Big[(q,q') \in \Phi \wedge q \xrightarrow{\tau} q_1 \Rightarrow (q_1,q') \in \Phi \vee \exists q'_1,q'_2.\big[q' \xrightarrow{\tau^*}\!\!\!\!\!\rightarrow\, q'_1 \xrightarrow{\tau} q'_2 \wedge (q,q'_1) \in \Phi \wedge (q_1,q'_2) \in \Phi\big]\Big]$.
- $\forall q,q',q'_1,x.\Big[(q,q') \in \Phi \wedge q' \xrightarrow{x} q'_1 \Rightarrow \exists q_1,q_2.\big[q \xrightarrow{\tau^*}\!\!\!\!\!\rightarrow\, q_1 \xrightarrow{x} q_2 \wedge (q_1,q') \in \Phi \wedge (q_2,q'_1) \in \Phi\big]\Big]$.
- $\forall q,q',q'_1.\Big[(q,q') \in \Phi \wedge q' \xrightarrow{\tau} q'_1 \Rightarrow (q,q'_1) \in \Phi \vee \exists q_1,q_2.\big[q \xrightarrow{\tau^*}\!\!\!\!\!\rightarrow\, q_1 \xrightarrow{\tau} q_2 \wedge (q_1,q') \in \Phi \wedge (q_2,q'_1) \in \Phi\big]\Big]$.

Let $q, q' \in Q$ be the initial states of processes $p, p' \in \mathbb{P}$, respectively. Two processes p and p' are said to be branching bisimilar (denoted as $p \leftrightarrows_{\mathrm{b}} p'$) iff there exists a branching bisimulation relation Φ such that there initial states q, q' are related, i.e. $(q, q') \in \Phi$. □

Note that in the absence of τ actions, branching bisimulation coincides with strong bisimulation. The phenomena of the occurrences of redundant silent steps can be formulated by the following notion of τ-inertness [9].

Definition 2.3. Let $p \in \mathbb{P}$ be an arbitrary process. A process p is said to be τ-*inert* with respect to $\leftrightarrows_{\mathrm{b}}$ iff for all states q of the transition system (generated by operational rules) of p it holds that $q \xrightarrow{\tau} q' \Rightarrow q \leftrightarrows_{\mathrm{b}} q'$ where, $q' \in Q$. □

The essence of the above definition is that an inert τ action does not affect the future choices of a process modulo branching bisimulation. In Section 5, we also show that an asynchronous closed loop system constructed from a synchronous closed loop system satisfying Definitions 5.2, 5.4 and 5.3 is always τ-inert with respect to $\leftrightarrows_{\mathrm{b}}$.

3 Supervisory control theory

In this section, we give a brief introduction to supervisory control theory and define its fundamental entities in our setup. The basic entity (a plant, or a supervisor, or a requirement) in the supervisory control theory is deterministic. Furthermore, the proof of main Theorem 5.10 requires the fact that a given synchronous closed loop system is also deterministic. Therefore, we now introduce the term deterministic process.

Definition 3.1. A process $p \in \mathbb{P}$ is called a *deterministic process* iff for all states q of the transition system (generated by the operational rules) of p and for all $x \in A$ it holds that $q \xrightarrow{x} q_1 \wedge q \xrightarrow{x} q_2 \Rightarrow q_1 \equiv q_2$ where, $q_1, q_2 \in Q$. □

In supervisory control theory, plants and supervisors are allowed to perform events that are divided into two disjoint subsets: *controllable* and *uncontrollable* events. The idea behind this partition is that the supervisor can enable or disable controllable events so that the closed loop behavior is equivalent to the requirement. The supervisor can observe but cannot influence uncontrollable events. In this paper, we follow the input-output interpretation [4] between a plant and its supervisor; wherein the uncontrollable events are outputs from a plant to a supervisor and the controllable events are outputs from a supervisor to a plant. Thus, processes that model plants or supervisors must have distinct (because of the above partition) input and output actions in its alphabet. Such processes are called input-output processes.

Definition 3.2. The set of input actions for an arbitrary process $p \in \mathbb{P}$ is denoted by $\alpha^{?}(p)$ and is defined as $\alpha^{?}(p) \triangleq \{?a \,|\, ?a \in \alpha(p)\}$. Similarly, the set of output actions (denoted by $\alpha^{!}(p)$) is defined as $\alpha^{!}(p) \triangleq \{!a \,|\, !a \in \alpha(p)\}$. A process p is called an *input-output process* iff

$$\alpha^{?}(p) \cap \alpha^{!}(p) = \emptyset \wedge \partial_I(p) \leftrightarrows_{\mathrm{b}} p \wedge \tau \notin \alpha(p)$$

where, $I = \{\mathbf{?}a \mid a \in Act\}$. □

The condition $\partial_I(p) \leftrightarrows_{\mathrm{b}} p$ ensures that an input-output process does not contain communicated actions in its alphabet. This is because bags are introduced to buffer both input and output events of an input-output process $p \in \mathbb{P}$. So if communicated actions are allowed in the specification of the process p then, the information whether the action $\mathbf{?}a$ is an input or an output action of the process p is unknown.

We now define the three basic entities in the supervisory control theory in our setup. A plant $P \in \mathbb{P}$ is a deterministic and an input-output process. Similarly, a supervisor is a deterministic and an input-output process. A requirement is a process specifying the legal interaction that should occur while the plant and its supervisor are interacting such that a required task (for which the supervisor is synthesised) is completed. Thus, a requirement is a deterministic process $R \in \mathbb{P}$ such that,

$$\partial_H(R) \leftrightarrows_{\mathrm{b}} R \wedge \tau \notin \alpha(R),$$

where $H = \{!a, ?a \mid a \in Act\}$. This condition ensures that a requirement process only contains communicated actions in its alphabet.

Now, we can state the control problem as follows: given a plant P and a requirement R, find a supervisor S such that,

$$\partial_H(P \parallel_\gamma S) \leftrightarrows_{\mathrm{b}} R.$$

In this paper, we are not interested in how this supervisor is computed and rather assume that we are provided with a solution to the above equation. The goal of this paper is then to find certain conditions on the given synchronous closed loop system such that it is desynchronisable. Note that in supervisory control theory the control problem is based on language equivalence, but branching bisimilarity coincides with language equivalence in the presence of determinism and in the absence of τ actions. However, we use branching bisimulation because the asynchronous closed loop systems as constructed in the next section are always nondeterministic. In brief, this cause of nondeterminism is due to the abstraction of interactions between a plant and the buffer.

4 From synchrony to asynchrony

In the previous section, we formally defined a plant P, a supervisor S and a requirement R in our setup. Now, we extend our setup in accordance with the architecture of Subsection 1.1, to model asynchronous communication by introducing two bags between a given plant and its supervisor; one bag that contains input actions of P and another one that contains output actions of P. Next we define a multiset and some operations over multisets that are necessary for the definition of a bag.

A multiset ξ over the set of communicated actions I is a tuple (I, κ) where $\kappa : I \to \mathbb{N}$ is the corresponding multiplicity function. We write the empty multiset as ε, which is defined as $(\emptyset, \kappa_0)$, where $\kappa_0 : \emptyset \to 0$ is the *zero function*.

Definition 4.1. Let $\xi = (I, \kappa)$ be a multiset over the set I.

- The predicate $\in'$ is used to denote an element that *belongs* to a multiset. It is defined as $?a \in' \xi \triangleq ?a \in I \wedge \kappa(?a) > 0$.
- The operator $\oplus$ is used to denote an *addition* of an element to a multiset. It is defined as $\xi \oplus ?a \triangleq (I', \kappa')$ where,
$$I' = \begin{cases} I, & \text{if } ?a \in I \\ I \cup \{?a\}, & \text{if } ?a \notin I \end{cases} \text{ and } \kappa'(x) = \begin{cases} \kappa(?a)+1, & \text{if } x = ?a \wedge ?a \in I \\ 1, & \text{if } x = ?a \wedge ?a \notin I \\ \kappa(x) & \text{if } x \neq ?a \wedge x \in I \end{cases}.$$
- The operator $\ominus$ is used to denote a *removal* of an element from a multiset. It is defined as $\xi \ominus ?a \triangleq (I', \kappa')$ where,
$$I' = \begin{cases} I, & \kappa(?a) > 1 \\ I \setminus \{?a\}, & \kappa(?a) = 1 \end{cases} \text{ and } \kappa'(x) = \begin{cases} \kappa(?a)-1, & \text{if } x = ?a \wedge \kappa(?a) > 0 \\ \kappa(x), & \text{if } x \neq ?a \wedge x \in I \end{cases}. \qquad \square$$

For each $x \in A$ we define a new element $\hat{x}$. Let $\hat{A}$ denote the set of new elements of the form $\hat{x}$. Similarly we assume that there exists auxiliary hidden and blocking sets $\hat{I}$ and $\hat{H}$, respectively.

Definition 4.2. (**Bag**). Let $n > 0$ be a natural number representing the size of a bag process. Let ε denote the empty multiset and ξ denote a multiset of communicated actions (i.e. the actions that are decorated with the symbol ⁉). Then a *bag process* over a set of actions $A_1 \subseteq A$ of size n is defined in the following way.

$$\begin{aligned}
B^n_{A_1}(\varepsilon,0) &= \sum_{?a\in A_1} ?\hat{a}.B^n_{A_1}(\varepsilon \oplus \text{⁉}a, 1) \ , \\
B^n_{A_1}(\xi,i) &= \sum_{\text{⁉}a\in'\xi} !\hat{a}.B^n_{A_1}(\xi \ominus \text{⁉}a, i-1) + \sum_{?b\in A_1} ?\hat{b}.B^n_{A_1}(\xi \oplus \text{⁉}b, i+1) \quad \text{for every } 0 < i < n \ , \\
B^n_{A_1}(\xi,n) &= \sum_{\text{⁉}a\in'\xi} !\hat{a}.B^n_{A_1}(\xi \ominus \text{⁉}a, n-1) \ .
\end{aligned}$$

□

The above definition is bounded with variable n that not only helps in modeling a realistic asynchronous implementation (as they contain buffers with finite memory). In contrast, it also aids in modeling an asynchronous implementation having buffers of infinite size, i.e., when $n = \infty$. Notation, we denote the two interleaving bags as,

$$B^{m,n}[\varepsilon,\varepsilon] \triangleq B^m_{A_1}(\varepsilon) \parallel B^n_{A_2}(\varepsilon)$$

where, $A_1 = \alpha^?(P), A_2 = \alpha^!(P)$ and $m > 0$ ($n > 0$) denotes the size of bag associated with input (output) actions of the plant P. Furthermore, the sets A_1 and A_2 denote the set of input and output actions of the plant P, respectively.

Next we formally define an abstraction scheme that implements the construction method M1. Informally, it decorates the interaction between a plant and the two interleaving bags with the symbol $\hat{\_}$, indicating such interactions are to be made hidden. We write the asynchronous closed loop system as $\tau_{\hat{I}}(\partial_{H\cup\hat{H}}(P \parallel_{\gamma'} B^{m,n}[\varepsilon,\varepsilon] \parallel_{\gamma'} S))$ (for some $m,n > 0$) constructed from its corresponding synchronous closed loop system $\partial_H(P \parallel_\gamma S)$ where,

- $\gamma' : \left(A \cup \hat{A}\right) \times \left(A \cup \hat{A}\right) \to \left(A \cup \hat{A}\right)$ is the modified communication function (or the abstraction scheme for method M1) defined in following way,
  $$\gamma'(!a,?\hat{a}) = \begin{cases} \text{⁉}\hat{a} & \text{if } !a \in \alpha^!(P) \\ \text{⁉}a & \text{if } !a \in \alpha^!(S) \end{cases} \quad \gamma'(!\hat{a},?a) = \begin{cases} \text{⁉}\hat{a} & \text{if } ?a \in \alpha^?(P) \\ \text{⁉}a & \text{if } ?a \in \alpha^?(S) \end{cases} \ .$$
  Intuitively, the communication function γ' with the operators $\tau_{\hat{I}}, \partial_{H\cup\hat{H}}$ ensures the interactions between the plant and the bag are invisible while the interactions between the supervisor and the bag are visible.

The rationale behind the choice of M1 is based on the observation that a transition system generated by a supervisor S is isomorphic to the corresponding synchronous closed loop system $\partial_H(P \parallel_\gamma S)$, modulo the difference in the type of action labels [6]. This is because in the synthesis of supervisors no transitions are introduced that a plant cannot execute. Moreover, the action labels in S will be decorated as either an input action ($?$) or an output action ($!$) while in $\partial_H(P \parallel_\gamma S)$ the same label will be decorated as a communicated action (⁉). Formally, this fact is equivalent to

$$\rho_f(S) \leftrightarrows \partial_H(P \parallel_\gamma S)$$

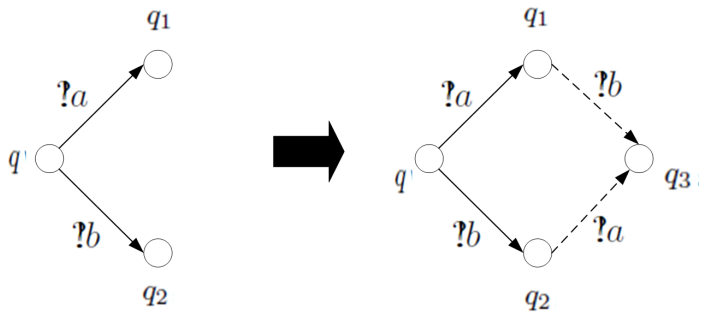


Figure 3: Diamond property.

where, ρ is the renaming operator from TCP [3] and $f : A \to A$ is a function that renames an input/output action to a communicated action, i.e., $\forall ?a, !a \in A.[f(!a) = f(?a) = \Updownarrow a]$. As a consequence, when one introduces bags and abstracts the interaction between plant and bags, the supervisor model remains unaffected. While in other abstraction schemes this is not the case. Thus, it is easier to study abstraction scheme M1 than other schemes.

5 Desynchronisable closed loop system

In the previous section, we have shown how to construct an asynchronous closed loop system from a given synchronous closed loop system. In general, the newly constructed asynchronous closed loop system will not be branching bisimilar to the given synchronous closed loop system. To this end, we introduce a special class of the synchronous closed loop system called desynchronisable closed loop system that are always branching bisimilar to their corresponding asynchronous closed loop systems. We then present sufficient conditions for desynchronisability.

Definition 5.1. Let $\partial_H(P \parallel_\gamma S)$ be a synchronous closed loop system and let m,n be any two nonzero natural numbers. Then, $\partial_H(P \parallel_\gamma S)$ is said to be *desynchronisable* with input and output buffers of size n and m (or in short desynchronisable closed loop system), respectively, if

$$\partial_H(P \parallel_\gamma S) \leftrightarrows_{\mathrm{b}} \tau_{\hat{I}}(\partial_{H\cup\hat{H}}(P \parallel_{\gamma'} B^{m,n}[\varepsilon,\varepsilon] \parallel_{\gamma'} S)) \ .$$

□

We now present three sufficient conditions for desynchronisability with buffers of arbitrary size. The objective of these conditions is the following. The conditions given in Definition 5.2 and Definition 5.3 prevent an asynchronous closed loop system from getting deadlocked. The condition in Definition 5.4 ensures that the silent steps introduced by the abstraction scheme are inert.

Definition 5.2. Let $\partial_H(P \parallel_\gamma S)$ be a synchronous closed loop system. Then, $\partial_H(P \parallel_\gamma S)$ is called *well posed* if there exists a binary relation $W \subseteq \mathbb{P} \times \mathbb{P}$ such that $(P,S) \in W$ and the following conditions are satisfied:

- $\forall\, !a, p, p', s.[(p,s) \in W \wedge p \xrightarrow{!a} p' \Rightarrow \exists s'.[s \xrightarrow{?a} s' \wedge (p',s') \in W]]$, and
- $\forall\, !a, p, s, s'.[(p,s) \in W \wedge s \xrightarrow{!a} s' \Rightarrow \exists p'.[p \xrightarrow{?a} p' \wedge (p',s') \in W]]$.

□

We now partition the set I into two disjoint non-empty subsets $I^?_P, I^!_P$ with respect to a plant process P as:

- $I^?_P \triangleq \{\Updownarrow a \mid \Updownarrow a \in I \wedge ?a \in \alpha^?(P)\}$.

- $I_P^! \triangleq \{ \text{?}a \mid \text{?}a \in I \wedge !a \in \alpha^!(P) \}$.

Definition 5.3. Let $\mu \in {I_P^?}^*$ and $\nu \in {I_P^!}^*$ be sequences in $I_P^?$ and $I_P^!$, respectively. Let $p \in \mathbb{P}$ be an arbitrary process and let $q \in Q$ be its initial state. We define the set of reachable states of p in the following way,

$$Reach(q) = \{ q' \mid \exists w \in A^*.[q \xrightarrow{w}\!\!\!\!\rightarrow q'] \} \ .$$

By the semantics of TCP we know that if the initial state of a process is of the structure $\partial_H(\_ \parallel_\gamma \_)$ then, all the reachable states will also be of the same structure. A synchronous closed loop system $\partial_H(P \parallel_\gamma S)$ is said to satisfy the *reordering* property iff both the following conditions are satisfied,

- $\forall p', p_2, s', \partial_H(p_1 \parallel_\gamma s_1) \in Reach(\partial_H(P \parallel_\gamma S)), \text{?}a \in I_P^?.$
  $\big[\partial_H(p_1 \parallel_\gamma s_1) \xrightarrow{\mu.\text{?}a}\!\!\!\!\rightarrow \partial_H(p' \parallel_\gamma s') \wedge p_1 \xrightarrow{?a} p_2 \Rightarrow \exists s_2.[\partial_H(p_1 \parallel_\gamma s_1) \xrightarrow{\text{?}a} \partial_H(p_2 \parallel_\gamma s_2)]\big]$
- $\forall p', s', s_2, \partial_H(p_1 \parallel_\gamma s_1) \in Reach(\partial_H(P \parallel_\gamma S)), \text{?}a \in I_P^!.$
  $\big[\partial_H(p_1 \parallel_\gamma s_1) \xrightarrow{\nu.\text{?}a}\!\!\!\!\rightarrow \partial_H(p' \parallel_\gamma s') \wedge s_1 \xrightarrow{?a} s_2 \Rightarrow \exists p_2.[\partial_H(p_1 \parallel_\gamma s_1) \xrightarrow{\text{?}a} \partial_H(p_2 \parallel_\gamma s_2)]\big].$

□

Definition 5.4. Let $q \in Q$ be an arbitrary state. Then, q is said to satisfy the *diamond* property iff the following condition (see Figure 3) holds

- $\forall \text{?}a, \text{?}b \in I, q_1, q_2.\Big[q \xrightarrow{\text{?}a} q_1 \wedge q \xrightarrow{\text{?}b} q_2 \wedge \text{?}a \neq \text{?}b \Rightarrow \exists q_3.[q_1 \xrightarrow{\text{?}b} q_3 \wedge q_2 \xrightarrow{\text{?}a} q_3]\Big] \ .$

A process p is said to satisfy the diamond property iff for all reachable states q' from q satisfy the diamond property, where q is the *initial state* of the process p. □

For a reader familiar with the concepts of *true concurrency* [17], the conditions given in Definition 3.1, 5.3 and 5.4 are similar to the axioms of asynchronous transition system. The formulation of these axioms is based on the definition of an independence relation, which is an irreflexive, symmetric relation on the set of actions A. However, the techniques for desynchronisability for such models are not investigated here, although it will be worthwhile to examine this research direction in the future. Note that in our approach we do not need an additional notion of the independence relation.

Next, we present the following main results of this paper.

- If an arbitrary synchronous closed loop system satisfy the conditions in Definitions 5.2, 5.3 and 5.4 then, it is a desynchronisable with buffers of arbitrary size.
- The transition system generated by an asynchronous closed loop system constructed from a synchronous closed loop system satisfying the conditions in Definitions 5.2, 5.3 and 5.4 is always τ-inert with respect to $\leftrightarrows_b$.

To prove the above statements, we first fix some notations and then prove some lemmas, which are necessary for the proof of main theorem.

We denote the contents of an arbitrary bag by the symbols ξ, ξ', i.e., ξ, ξ' are of the form (I_0, κ_0) and (I_1, κ_1) respectively, where $I_0, I_1 \subseteq I$. The contents of the bag attached to input actions of P is denoted by μ, i.e., μ is of the form (I_μ, κ_μ) where $I_\mu \subseteq I_P^?$. Similarly the contents of the bag attached to output actions of P is denoted by ν, i.e., ν is of the form (I_ν, κ_ν) where $I_\nu \subseteq I_P^!$. For an arbitrary multiset ξ, we define a sequence (denoted as $\vec{\xi}$) over ξ as,

$$\vec{\xi} \triangleq < x_1.x_2.\cdots >$$

such that $\#(x_i, \vec{\xi}) = \kappa(x_i)$, where # is a function that returns the maximum number of occurrences of x_i in $\vec{\xi}$ for some $i > 0$. For example consider a multiset $\xi = \{?a, ?a, ?b, ?b\}$. Then a possible sequence $\vec{\xi}$ over the given ξ can be of the form $< ?a.?b.?a.?b >$. Let $f_i : I^* \to H^*$ be the function defined as $f_i(?a.\vec{\xi}) = ?a.f_i(\vec{\xi})$. Similarly, let $f_o : I^* \to H^*$ be the function defined as $f_o(?a.\vec{\xi}) = !a.f_o(\vec{\xi})$.

Proposition 5.5. Given a trace $\partial_H(P \parallel_\gamma S) \xrightarrow{\vec{\mu}} \partial_H(p_1 \parallel_\gamma s_1)$, we find using the above function f_i and semantics of $\parallel_\gamma$ that $P \xrightarrow{f_i(\vec{\mu})} p_1 \wedge S \xrightarrow{f_o(\vec{\mu})} s_1$.

Proposition 5.6. Similarly, given a trace $\partial_H(P \parallel_\gamma S) \xrightarrow{\vec{\nu}} \partial_H(p_1 \parallel_\gamma s_1)$, we conclude that $P \xrightarrow{f_o(\vec{\nu})} p_1 \wedge S \xrightarrow{f_i(\vec{\nu})} s_1$.

The following lemma is a generalisation of Definition 5.4. It states that if two different states q_1, q_2 are reachable from a state q_0, then there exists a state q_3 reachable from q_1 and q_2 such that, the trace between q_0, q_1 and the trace between q_0, q_2 commute.

Lemma 5.7 (Generalised diamond property). *Let $\partial_H(P \parallel_\gamma S)$ be an arbitrary synchronous closed loop system satisfying the conditions in Definitions 5.2, 5.3 and 5.4. If $\partial_H(P \parallel_\gamma S) \xrightarrow{\vec{\xi}} \partial_H(p_1 \parallel_\gamma s_1) \wedge \partial_H(P \parallel_\gamma S) \xrightarrow{\vec{\xi}'} \partial_H(p_2 \parallel_\gamma s_2)$ then,*

$$\exists p_3, s_3.[\partial_H(p_1 \parallel_\gamma s_1) \xrightarrow{\vec{\xi}'} \partial_H(p_3 \parallel_\gamma s_3) \wedge \partial_H(p_2 \parallel_\gamma s_2) \xrightarrow{\vec{\xi}} \partial_H(p_3 \parallel_\gamma s_3)].$$

The following Lemmas 5.8, 5.9 are the results (See [5] for the proofs) obtained by direct instantiation of reordering property (Definition 5.3) and generalised diamond property (Lemma 5.7).

Lemma 5.8. *Let $\partial_H(P \parallel_\gamma S)$ be a synchronous closed loop system satisfying the conditions in Definitions 5.2, 5.3 and 5.4. Suppose $?a \in I_P^? \wedge \partial_H(P \parallel_\gamma S) \xrightarrow{\vec{\mu}.?a} \partial_H(p_2 \parallel_\gamma s_2) \wedge P \xrightarrow{?a} p_1$ then,*

$$\exists s_1. \left[\partial_H(P \parallel_\gamma S) \xrightarrow{?a} \partial_H(p_1 \parallel_\gamma s_1) \xrightarrow{\vec{\mu}} \partial_H(p_2 \parallel_\gamma s_2)\right].$$

Lemma 5.9. *Let $\partial_H(P \parallel_\gamma S)$ be a synchronous closed loop system satisfying the conditions in Definitions 5.2, 5.3 and 5.4. Suppose $?a \in I_P^! \wedge \partial_H(P \parallel_\gamma S) \xrightarrow{\vec{\nu}.?a} \partial_H(p_3 \parallel_\gamma s_3) \wedge S \xrightarrow{?a} s_1$ then,*

$$\exists p_1. \left[\partial_H(P \parallel_\gamma S) \xrightarrow{?a} \partial_H(p_1 \parallel_\gamma s_1) \xrightarrow{\vec{\nu}} \partial_H(p_3 \parallel_\gamma s_3)\right].$$

We now pose the main theorem of this paper which proves the following statement. If the given synchronous closed system satisfies the conditions in Definition 5.2, 5.3 and 5.4, then it is desynchronisable independent of the size of the buffers introduced between the given plant and its supervisor.

Theorem 5.10. *Let $\partial_H(P \parallel_\gamma S)$ be an arbitrary synchronous closed loop system satisfying the conditions in Definitions 5.2, 5.3 and 5.4. Then for any $m, n > 0$ we have,*

$$\partial_H(P \parallel_\gamma S) \leftrightarrows_b \tau_{\hat{I}}(\partial_{H \cup \hat{H}}(P \parallel_\gamma B^{m,n}[\varepsilon, \varepsilon] \parallel_\gamma S)).$$

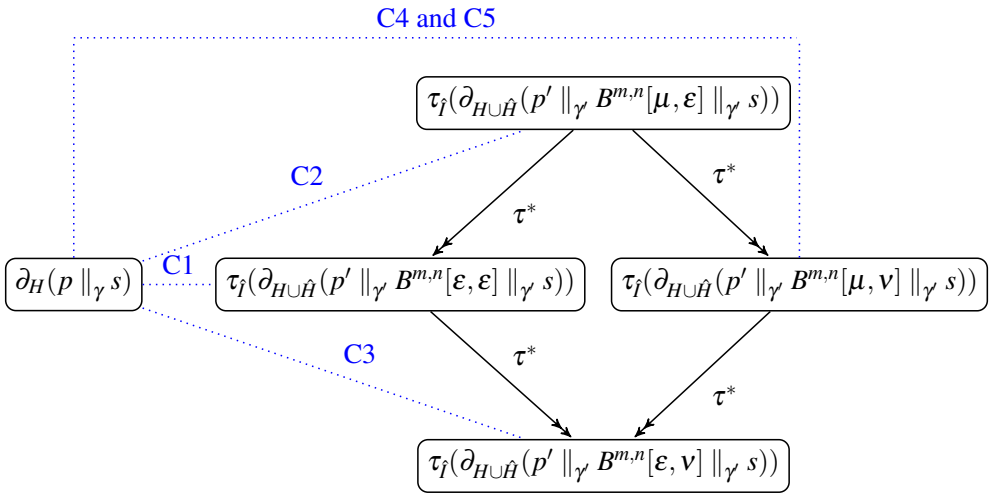


Figure 4: Illustration of relation Φ.

Proof. Let p, p', s be free process variables. Let μ, ν be two free variables representing the contents of an input and an output bag of P, respectively. Then, define a relation Φ as follows.

$$\begin{aligned}
\Phi \triangleq \{(\partial_H(p \parallel_\gamma s), \tau_{\hat{I}}(\partial_{H\cup\hat{H}}(p' \parallel_{\gamma'} B^{m,n}[\mu,\nu] \parallel_{\gamma'} s))) \mid \\
\Big(p' = p \wedge \mu = \varepsilon \wedge \nu = \varepsilon\Big) \bigvee \quad \text{[C1]} \\
\Big(\mu = \varepsilon \wedge \exists s'.\big[\partial_H(p \parallel_\gamma s) \xrightarrow{\vec{\nu}}\!\!\!\!\rightarrow \partial_H(p' \parallel_\gamma s')\big]\Big) \bigvee \quad \text{[C2]} \\
\Big(\nu = \varepsilon \wedge \exists s'.\big[\partial_H(p' \parallel_\gamma s') \xrightarrow{\vec{\mu}}\!\!\!\!\rightarrow \partial_H(p \parallel_\gamma s)\big]\Big) \bigvee \quad \text{[C3]} \\
\Big(\exists p'', s', s''.\big[\partial_H(p' \parallel_\gamma s') \xleftarrow{\vec{\nu}} \partial_H(p'' \parallel_\gamma s'') \xrightarrow{\vec{\mu}}\!\!\!\!\rightarrow \partial_H(p \parallel_\gamma s)\big]\Big) \bigvee \quad \text{[C4]} \\
\Big(\exists p'', s', s''.\big[\partial_H(p \parallel_\gamma s) \xrightarrow{\vec{\nu}}\!\!\!\!\rightarrow \partial_H(p'' \parallel_\gamma s'') \xleftarrow{\vec{\mu}} \partial_H(p' \parallel_\gamma s')\big]\Big)\ \}. \quad \text{[C5]}
\end{aligned}$$

Note that the above conditions C1, C2, C3, C4 and C5 are independent of n, m. The proof of the theorem is based on showing that Φ is a witnessing branching bisimulation relation. The intuition behind the definition of Φ is the following. A state $\partial_H(p \parallel_\gamma s)$ in a synchronous closed loop system is related to those states in an asynchronous closed loop system that contain the same supervisor state s. The Φ relation between two states is indicated by dotted lines in Figure 4. The complete proof requires a lot of case distinction and can be found in [5]. Here we discuss the different cases that are present in the proof and give the list of lemmas that are applied in each case. Let q_c, q_a be the initial states of the processes $\partial_H(p \parallel_\gamma s)$ and $\tau_{\hat{I}}(\partial_{H\cup\hat{H}}(p' \parallel_{\gamma'} B^{m,n}[\mu,\nu] \parallel_{\gamma'} s))$, respectively. From the definition of branching bisimilarity we need to show the following four transfer conditions:

1. $\forall\, ?a, q_c, q_c', q_a.[q_c \xrightarrow{?a} q_c' \wedge (q_c, q_a) \in \Phi \Rightarrow \exists q_a', q_a''.[q_a \xrightarrow{\tau^*}\!\!\!\!\rightarrow q_a' \xrightarrow{?a} q_a'' \wedge (q_c, q_a'), (q_c', q_a'') \in \Phi]]$.
2. $\forall q_c, q_c', q_a.[q_c \xrightarrow{\tau} q_c' \wedge (q_c, q_a) \in \Phi \Rightarrow (q_c', q_a) \in \Phi \vee \exists q_a', q_a''.[q_a \xrightarrow{\tau^*}\!\!\!\!\rightarrow q_a' \xrightarrow{\tau} q_a'' \wedge (q_c, q_a'), (q_c', q_a'') \in \Phi]]$.
3. $\forall q_c, q_a, q_a'.[q_a \xrightarrow{\tau} q_a' \wedge (q_c, q_a) \in \Phi \Rightarrow (q_a', q_c) \in \Phi \vee \exists q_c', q_c''.[q_c \xrightarrow{\tau^*} q_c' \xrightarrow{\tau} q_c'' \wedge (q_c', q_a), (q_c'', q_a') \in \Phi]]$.

4. $\forall ?a, q_c, q_a, q'_a.[q_a \xrightarrow{?a} q'_a \wedge (q_c, q_a) \in \Phi \Rightarrow \exists q'_c, q''_c.[q_c \xrightarrow{\tau^*} q'_c \xrightarrow{?a} q''_c \wedge (q'_c, q_a), (q''_c, q'_a) \in \Phi]]$.

Since the synchronous closed loop system does not contain τ actions in its alphabet, there are following three effects on the above transfer conditions.

- The above condition 2 will be vacuously satisfied.
- The condition 3 will be reduced to the simpler form,

$$\forall q_c, q_a, q'_a.\big[q_a \xrightarrow{\tau} q'_a \wedge (q_c, q_a) \in \Phi \Rightarrow (q_c, q'_a) \in \Phi\big].$$

- And similarly condition 4 will be reduced to:

$$\forall ?a, q_c, q_a, q'_a.[q_a \xrightarrow{?a} q'_a \wedge (q_c, q_a) \in \Phi \Rightarrow \exists q'_c.[q_c \xrightarrow{?a} q'_c \wedge (q'_c, q'_a) \in \Phi]].$$

But to show that these conditions hold, we need to know whether an action label $?a$ occurring in each condition is either an input or output action with respect to P, i.e. $?a \in I^?_P$ or $?a \in I^!_P$. Thus, we get six transfer conditions in total that are shown in Table 1. Furthermore, for each case we apply case distinction based on the structure of μ and ν. In each subcase we use C1, C2, C3, C4, and C5 (the conditions from the definition of Φ) to determine the relation between free process variables p, p' and then prove the conclusion as shown in Table 1. The notation $\tau = \tau_{?a}(?a)$ is used to denote that the τ action is a result of abstraction of the communicated action $?a$.

Table 1: Proof structure of Theorem 5.10.

Case No.	Hypothesis	Conclusion
T1	$q_c \xrightarrow{?a} q'_c \wedge (q_c, q_a) \in \Phi \wedge ?a \in I^?_P$.	$\exists q'_a, q''_a.[q_a \xrightarrow{\tau^*} q'_a \xrightarrow{?a} q''_a \wedge (q_c, q'_a), (q'_c, q''_a) \in \Phi.]$
T2	$q_c \xrightarrow{?a} q'_c \wedge (q_c, q_a) \in \Phi \wedge ?a \in I^!_P$.	$\exists q'_a, q''_a.[q_a \xrightarrow{\tau^*} q'_a \xrightarrow{?a} q''_a \wedge (q_c, q'_a), (q'_c, q''_a) \in \Phi.]$
T3	$q_a \xrightarrow{\tau} q'_a \wedge (q_c, q_a) \in \Phi \wedge ?a \in I^?_P$ $\wedge\ \tau = \tau_{?a}(?a)$.	$(q_c, q'_a) \in \Phi$.
T4	$q_a \xrightarrow{\tau} q'_a \wedge (q_c, q_a) \in \Phi \wedge ?a \in I^!_P$ $\wedge\ \tau = \tau_{?a}(?a)$.	$(q_c, q'_a) \in \Phi$.
T5	$q_a \xrightarrow{?a} q'_a \wedge (q_c, q_a) \in \Phi \wedge ?a \in I^?_P$.	$\exists q'_c.[q_c \xrightarrow{?a} q'_c \wedge (q'_c, q'_a) \in \Phi.]$
T6	$q_a \xrightarrow{?a} q'_a \wedge (q_c, q_a) \in \Phi \wedge ?a \in I^!_P$.	$\exists q'_c.[q_c \xrightarrow{?a} q'_c \wedge (q'_c, q'_a) \in \Phi.]$

In Table 2 we present the list of lemmas required to prove each case.

□

In hindsight, what we have actually proven is that all τ actions generated by the abstraction scheme are τ-inert with respect to $\leftrightarrows_b$. The following corollary states this fact.

Corollary 5.11. *Let q_c be a process of the form $\partial_H(P \parallel_\gamma S)$. And let q_a be an asynchronous closed loop system of the form $\tau_{\hat{I}}(\partial_{H\cup\hat{H}}(P' \parallel_{\gamma'} B^{m,n}[\mu,\nu] \parallel_{\gamma'} S))$ such that $(q_c, q_a) \in \Phi$. Then,*

$$\forall q_c, q_a, q'_a.\big[(q_c, q_a) \in \Phi \wedge q_a \xrightarrow{\tau} q'_a \Rightarrow q_a \leftrightarrows_b q'_a\big].$$

Case No.	List of lemmas
T1	Lemma 5.7
T2	Lemma 5.7 and Proposition 5.5
T3	Lemma 5.7 and Lemma 5.8
T4	Lemma 5.7
T5	Lemma 5.7 and Lemma 5.9

Table 2: List of lemmas applied in each case.

6 Conclusions and future work

In this paper, we presented sufficient conditions for desynchronisability in a process algebraic setting and showed that an asynchronous implementation using bags (of arbitrary size) is a refinement of the synchronous closed loop system satisfying these conditions. The prominent features of our work can be summarised in the following main points:

- We solve a refinement problem instead of a supervisory control problem, and do not compute a new supervisor in the presence of buffers, as done in [4, 18]. Our approach is *intended* to be computationally cheaper than the one developed in [4, 18], however this conjecture needs to be verified by analysing the complexities associated with the conditions presented here. In particular, we conjecture that supervisory control theory always results in synchronous closed loop systems, which are well-posed (Definition 5.2), but the other conditions, (Definition 5.3 and Definition 5.4), are not likely to be attained so easily.
- We present our conditions for desynchronisability over the components of a synchronous closed loop system conjointly, in contrast with [8], where the check for the foam rubber wrapper principle on the two components was applied separately. Note the sender domination property from [8] is equivalent to the well posed condition (Definition 5.2). However, the two approaches are incomparable because in [8] the construction method M3 was studied while in this paper the construction method M1 is studied.
- We use branching bisimulation equivalence instead of failure equivalence that was adopted in [8]. As a consequence our techniques are applicable to all the weak equivalences in 'van Glabbeek spectrum' [16] (including failure equivalence). The branching bisimulation is the preferred equivalence in TCP process algebra under the presence of τ action [3]. Furthermore, the conditions (well posedness and diamond property) given here are similar to the ones mentioned in [8], where desynchronisability was characterised modulo failure equivalence. Thus, we conjecture that achieving desynchronisability for weaker equivalences will not lead to weaker sufficient conditions.

A question that was not treated in this paper, is whether the conditions we posed are in fact reasonable for industrial applications. This may become clear in the near future, when we study the case studies involved with supervisory control theory in the context of MULTIFORM project [1] with the language CIF [2]. The authors of CIF are currently developing techniques that will incorporate supervisory control theory and model based engineering into a single framework, thus making it suitable for the design of industrial applications. In particular, the elevator case study and the toy example, which were desynchronisable in [6] using the construction method M1, satisfy our conditions.

Another question is whether the conditions given are actually necessary for desynchronisability modulo branching bisimilarity. Formally speaking, they are not, and counter-examples have been found although we do not give them here. We anticipate that the diamond property (Definition 5.4) can be further

weakened. In particular, if the actions $?a, ?b \in I_P^?$ are enabled at a state q then it may not be necessary for the traces $?a.?b$ and $?b.?a$ to commute.

Lastly, the research performed in this paper can of course be repeated for different architectures. One might study whether wires or queues can be used instead of bags, or study different abstraction schemes, or try to study the conditions for desynchronisability by focusing on other notions of weak equivalences.

7 Acknowledgements

The authors would like to thank the reviewers for their valuable critical remarks on the earlier draft of this paper and pointing out that the sender domination property and the well posed property are equivalent. The authors would also like to thank Jos Baeten, Koos Rooda, Bert van Beek, Rong Su, Jasen Markowski, Damian Nadales, and Mihaly Petreczky for various discussions and clarification about this work.

This work has been performed as part of the "Integrated Multi-formalism Tool Support for the Design of Networked Embedded Control Systems" (MULTIFORM) project, supported by the Seventh Research Framework Programme of the European Commission (Grant agreement number: INFSO-ICT-224249).

References

[1] Integrated multi-formalism tool support for the design of networked embedded control systems : Multiform. `http://cms.multiform.bci.tu-dortmund.de/`.

[2] D. E. Nadales Agut, D. A. van Beek, R. R. H. Schiffelers, D. Hendriks, and J. E. Rooda. Abstract syntax and formal semantics of the CIF. Technical report, Eindhoven University of Technology, October 2009.

[3] J. C. M. Baeten, T. Basten, and M. Reniers. *Process Algebra: Equational Theories of Communicating Processes*. Cambridge University Press, 2009.

[4] S. Balemi. *Control of Discrete Event Systems: Theory And Application*. PhD thesis, Swiss Federal Institute of Technology, Automatic Control Laboratory, ETH Zurich, May 1992.

[5] H. Beohar and P. J. L. Cuijpers. A theory of desynchronisable closed loop systems. To appear as Technical report, Eindhoven university of technology, 2010. `http://www.win.tue.nl/~pcuijper/pages/publications.html`.

[6] H. Beohar, P. J. L. Cuijpers, and J. C. M. Baeten. Design of asynchronous supervisors. abs/0910.0868, 2009. `http://arxiv.org/abs/0910.0868`.

[7] M. Fabian and A. Hellgren. PLC-based implementation of supervisory control for discrete event systems. *Proceedings of the 37th IEEE Conference on Decision and Control*, 3:3305–3310, 1998.

[8] C. Fischer and W. Janssen. Synchronous development of asynchronous systems. In Ugo Montanari and Vladimiro Sassone, editors, *Proceedings of CONCUR'96*, volume 1119 of *Lecture Notes in Computer Science*, pages 735–750. Springer-Verlag, 1996.

[9] J. F. Groote and M. P. A. Sellink. Confluence for process verification. *Theor. Comput. Sci.*, 170 (1-2):47–81, 1996.

[10] He Jifeng, M. B. Josephs, and C. A. R. Hoare. A theory of synchrony and asynchrony. In M. Broy and C. B. Jones, editors, *Programming Concepts and Methods*, pages 459–479, 1990.

[11] H. K. Kapoor and M. B. Josephs. Modelling and verification of delay-insensitive circuits using CCS and the concurrency workbench. volume 89, pages 293–296, 2004.

[12] M. Mousavi, P. Le Guernic, J.-P. Talpin, S. K. Shukla, and T. Basten. Modeling and validating globally asynchronous design in synchronous frameworks. In *Proceedings of the Conference on Design Automation and Test in Europe*, pages 384–389. IEEE Computer Society Press, 2003.

[13] G. D. Plotkin. A Structural Approach to Operational Semantics. Technical Report DAIMI FN-19, University of Aarhus, 1981.

[14] P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. *SIAM Journal on Control and Optimization*, 25(1):206–230, 1987.

[15] J.T. Udding. *Classification and Composition of Delay-Insensitive Circuits*. PhD thesis, Eindhoven University of Technology, Eindhoven, 1984.

[16] R. J. v. Glabbeek. *Comparative concurrency semantics and refinement of actions*. Ph.D. thesis, CWI, Amsterdam, 1990.

[17] G. Winskel and M. Nielsen. Models for concurrency. In *Handbook of Logic in Computer Science*, pages 1–148. Oxford University Press, 1995.

[18] S. Xu and R. Kumar. Asynchronous implementation of synchronous discrete event control. pages 181 –186, May 2008. doi: 10.1109/WODES.2008.4605942.

A Operational semantics of TCP

In this section, we give the SOS rules for the operators used in this paper. Note that the rules for symmetric case (in the context of binary operators) are not given.

$$1\ \frac{}{x.p \xrightarrow{x} p} \qquad 2\ \frac{p \xrightarrow{x} p'}{\begin{array}{c} p+q \xrightarrow{x} p' \\ q+p \xrightarrow{x} p' \end{array}} \qquad 3\ \frac{p \xrightarrow{x} p'}{\begin{array}{c} p \parallel_\gamma q \xrightarrow{x} p' \parallel_\gamma q \\ q \parallel_\gamma p \xrightarrow{x} q \parallel_\gamma p' \end{array}} \qquad 4\ \frac{p \xrightarrow{x} p', q \xrightarrow{x'} q', \gamma(x,x') = x''}{\begin{array}{c} p \parallel_\gamma q \xrightarrow{x''} p' \parallel_\gamma q' \\ q \parallel_\gamma p \xrightarrow{x''} q' \parallel_\gamma p' \end{array}}$$

$$5\ \frac{p \xrightarrow{x} p', x \notin H}{\partial_H(p) \xrightarrow{x} \partial_H(p')} \qquad 6\ \frac{p \xrightarrow{x} p', x \notin I}{\tau_I(p) \xrightarrow{x} \tau_I(p)} \qquad 7\ \frac{p \xrightarrow{x} p', x \in I}{\tau_I(p) \xrightarrow{\tau} \tau_I(p)}$$

$$8\ \frac{t_0 \xrightarrow{x} p, X_0 = t_0}{X_0 \xrightarrow{x} p} \qquad 9\ \frac{p \xrightarrow{x} p', f : A \to A}{\rho_f(p) \xrightarrow{f(x)} \rho_f(p')}$$

Definition A.1. The alphabet of a process p, written as $\alpha(p)$, is the set of atomic actions that it can perform. It is defined for the following closed terms.

$$\begin{aligned} \alpha(\mathbf{0}) &= \emptyset \\ \alpha(x.p) &= \{x\} \cup \alpha(p) \\ \alpha(\tau.p) &= \alpha(p) \\ \alpha(p+q) &= \alpha(p) \cup \alpha(q) \end{aligned}$$

Note that α is not defined explicitly for the operators $\parallel_\gamma, \partial_H, \tau_I$ because these operators can be eliminated (See the corresponding elimination theorems in [3]). □